National Aeronautics and
Space Administration

NOT MEASUREMENT
SENSITIVE

NASA-STD-8719.13A
SEPTEMBER 15, 1997

REPLACES NSS 1740.13 DATED FEBRUARY 1996

SOFTWARE SAFETY

NASA TECHNICAL STANDARD

PREFACE

This standard provides a methodology for software safety in NASA programs. It describes the activities necessary to ensure that safety is designed into software that is acquired or developed by NASA. All Program/Project Managers are to assess the inherent safety risk of the software in their individual programs and are encouraged to tailor their software safety activity accordingly within the framework of this standard.

This standard expands on the requirements of NMI 2410.10, "NASA Software Management Assurance and Engineering Policy," NHB 1700.1(V1), "NASA Safety Policy and Requirements Document"; and NASA-STD-2201-93, "Software Assurance Standard."

If a mandatory (shall, will, must) requirement cannot be met, a deviation/waiver package shall be prepared according to NHB 1700.1 (V1). NASA deviations/waivers to requirements in this document shall be approved, as a minimum, by the Installation or program safety official. The Office of Safety and Mission Assurance shall be notified semiannually of all variances to safety requirements contained in this standard and approved at the Installation or program levels.

Comments and questions concerning the contents of this publication should be referred to the National Aeronautics and Space Administration Headquarters, Director, Safety and Risk Management Division, Office of the Associate Administrator for Safety and Mission Assurance, Washington, DC 20546.

Original Signed By

Frederick Gregory
Associate Administrator for
Safety and Mission Assurance

EFFECTIVE DATE:
Feb 12, 1996

This page intentionally left blank.

RECORD OF CHANGES

Change No.	Date	Title or Brief Description	Entered By	Date Entered
1	9/97	Headquarters documentation numbering	White	9/10/97
2	9/97	Paragraph 3.1(e); assess scope and level of IV&V	White	9/10/97

This page intentionally left blank.

TABLE OF CONTENTS

1.0 INTRODUCTION

1.1 SCOPE

This standard provides a methodology for software safety in NASA programs.

1.2 PURPOSE

The purpose of this standard is to provide requirements to implement a systematic approach to software safety as an integral part of the overall system safety programs. It describes the activities necessary to ensure that safety is designed into software that is acquired or developed by NASA and that safety is maintained throughout the software life cycle.

1.3 APPLICABILITY

This standard is appropriate for application to software acquired or developed by NASA that is used as a part of a system that possesses the potential of directly or indirectly causing harm to humans or damage to property external to the system. When software is acquired by NASA, this standard applies to the level specified in contract clauses or memoranda of understanding. When software is developed by NASA, this standard applies to the level specified in the program plan, software management plan, or other controlling document.

This standard is intended to be applied only to specific software elements identified by a system hazard analysis as that software which could cause or contribute to the system reaching a specific hazardous state; or which is intended to detect or take corrective action if the system reaches a specific hazardous state; or which is intended to mitigate damage if an accident occurs.

1.3.1 GOVERNMENT FURNISHED EQUIPMENT (GFE), REUSED, AND PURCHASED SOFTWARE

For systems where use of this standard is required, it shall be applied to government furnished software, purchased software (including commercial-off-the-shelf (COTS) software), and any other reused software in the system. In the event that some of the analyses required by this document are not feasible due to the nature of the software and documentation, the developer is responsible for securing a waiver from the NASA acquirer of the system.

1.3.2 FIRMWARE

For the purpose of this standard, firmware shall be treated as software.

1.4 TAILORING

Tailoring of this standards requirements is to be done by program/project/task management in consultation with each respective NASA Centers Safety and Mission Assurance organization. The tailoring effort shall include definition of the acceptable level of risk, which software is to be considered safety-critical, and whether the level of safety risk associated with the software requires formal safety certification.

This page intentionally left blank.

2.0 REFERENCES

2.1 REFERENCED DOCUMENTS

The following references were used in the generation of this Standard.

	NASA Software Acquisition Life Cycle, SMAP/Version 4.0, 1989
DoD-STD-2167A	Military Standard, Defense System Software Development, February 29, 1988
MIL-STD-882B, NOTICE 1	System Safety Program Requirements, July 1, 1987
MIL-STD-882C	System Safety Program Requirements, January 19, 1993
NASA-GB-A302	Software Formal Inspections Guidebook, August 1993
NASA-STD-2100-91	NASA Software Documentation Standard Software Engineering Program, July 29, 1991
NASA-STD-2201-93	Software Assurance Standard, November 10, 1992
NASA-STD-2202-93	Software Inspection Standard, April 1993
NHB 1700.1(V1-B)	NASA Safety Policy and Requirements Document, June 1993
NMI 2410.10B	NASA Software Management Assurance and Engineering Policy, April 20, 1993
SMAP-GB-A201	Software Assurance Guidebook, September 1989
SMAP-GB-A301	Software Quality Assurance Audits Guidebooks, November 1990

2.2 GLOSSARY

A glossary of software safety-related terms is provided in Appendix A.

2.3 ABBREVIATIONS AND ACRONYMS

A list of abbreviations and acronyms is provided in Appendix B.

This page intentionally left blank.

3.0 REQUIREMENTS

3.1 GENERAL

The purpose of the software safety activities is to ensure that software does not cause or contribute to a system reaching a hazardous state; that it does not fail to detect or take corrective action if the system reaches a hazardous state; and that it does not fail to mitigate damage if an accident occurs.

The software safety process shall:

a. Ensure that the system/subsystem safety analyses identify which software is safety-critical. Any software that has the potential to cause a hazard or is required to support control of a hazard, as identified by safety analyses, is safety-critical software.

b. Ensure that the system/subsystem safety analyses clearly identify the key inputs into the software requirements specification (e.g., identification of hazardous commands, limits, interrelationship of limits, sequence of events, timing constraints, voting logic, failure tolerance, etc.).

c. Ensure that the development of the software requirements specification includes the software safety requirements that have been identified by software safety analysis.

d. Ensure that the software design and implementation properly incorporate the software safety requirements.

e. Ensure that the appropriate verification and validation requirements are established to ensure proper implementation of the software safety requirements. This explicitly includes an assessment of the scope and level of IV&V to be planned and implemented based on the level of criticality and risk of the software application. A statement will be made in either the program/project plan or the software development plan as to the level of IV&V to be accomplished.

f. Ensure that test plans and procedures will satisfy the intent of the software safety verification requirements.

g. Ensure that the results of the software safety verification effort are satisfactory.

3.2 SYSTEM SAFETY ANALYSES

Preliminary system safety analyses (e.g., Preliminary Hazard Analysis (PHA)), conducted during the system requirements phase when the role of software is being defined, begin to identify the hazards associated with a particular design concept and/or operation. These preliminary analyses and subsequent system and software safety analyses identify when software is a potential cause of a hazard or will be used to support the control of a hazard. This software shall be classified as safety-critical and shall be subjected to software safety analysis. Safety-critical software is typically software that if not performed or is performed incorrectly, inadvertently, or out of sequence could result in a hazard or allow a hazardous condition to exist, such as (1) software that exercises direct command and control over potentially hazardous functions and/or hardware, (2) software that monitors critical hardware components, and/or (3) software that monitors the system for possible critical conditions and/or states.

The system safety analyses are the first place to identify software safety requirements necessary to support the development of the software requirements specification. These requirements shall be provided to the developer for inclusion into the software requirements document. Some examples of software safety requirements include limits (e.g., redlines, boundary values), sequence of events, timing constraints, interrelationship of limits, voting logic, hazardous hardware failure recognition, failure tolerance, caution and warning interfaces, hazardous commands, etc.

The system safety analyses continue throughout the project life cycle. The software safety analysis process needs to continue to review the results of the systems analyses to assure that changes and findings at the system level are incorporated into the software as necessary. In addition, the software safety analyses provide input to the system safety analyses. The software safety analyses are a special portion of the overall system safety analyses and are not conducted in isolation.

3.3 SOFTWARE SAFETY

Software safety shall be an integral part of the overall system safety and software development efforts. It is the objective of the software safety effort to ensure that safety is considered throughout the software life cycle. Therefore, software safety activities take place in every phase of the system and software development life cycle beginning as early as the concept phase and on through to operations and maintenance. Up-front participation, analyses, and subsequent reporting of safety problems found during the software development life cycle facilitates timely and less costly solutions.

Software safety requires a coordinated effort among all organizations involved in the development of NASA software. This includes Program Managers, hardware and software designers, safety analysts, quality assurance, and operations personnel. Those conducting the Software Safety effort shall also interface with personnel from disciplines such as reliability, security, Independent Verification and Validation (IV&V) (when available), and human factors.

3.3.1 SOFTWARE SAFETY TASKS BY LIFE CYCLE PHASE

The following subsections describe which software safety tasks are appropriate for each software development life cycle phase, using the waterfall life cycle as the primary life cycle methodology. Many of the safety analyses are iterative, taking place during the software development life cycle (i.e., the results from one phase feed the analyses of the next). As the detail of the project evolves, so does the maturity of the safety analysis.

3.3.1.1 SOFTWARE SAFETY PLANNING

Software safety planning shall be done for each software acquisition to which this standard is applied. The planning shall be documented in the required Software Management Plan or Safety Management Plan. The Software Management Plan shall be formatted in accordance with the Software Management Plan DID in NASA-STD-2100-91 "NASA Software Documentation Standard." The Safety Management Plan shall be documented in accordance with Section 306 of NHB 1700.1 (V1-B). If the safety planning is documented in multiple plans, each plan shall include a cross-reference to the safety activities in the remaining plans. The plan shall describe how the activities specified by this standard will be implemented. The plan shall specify the activities to be carried out, the schedule on which they will be implemented, and the products that will result. The plan shall address the interrelationships among system safety analysis, software safety analysis, and the software development efforts. The plan shall specifically address the mechanism by which safety-critical requirements are generated, implemented, tracked, and verified.

3.3.1.2 SOFTWARE REQUIREMENTS SPECIFICATION DEVELOPMENT

During the software requirements specification development, the system-level requirements allocated to software are analyzed and documented as software requirements. Test planning is begun, with a method for verifying each requirement identified and included in a preliminary test plan. Risks are identified and risk management control mechanisms are established. Two software safety tasks shall be performed in this phase of the software development life cycle:

a. Development of software safety requirements.

b. Analysis of the software requirements for potential hazards.

The successful development of safety requirements for the software requirements specification is essential to developing safe software and allows for safety to be built into the software early in the life cycle while it is relatively inexpensive. The process of development of software safety requirements involves analysis of system safety requirements, hardware, software, and user interfaces, and areas of system performance where software is a potential cause of a hazard or supports control of a hazard identified by the system safety analyses. These system requirements, interfaces, and areas of performance shall be analyzed to develop the software requirements necessary to ensure that the related hazards are properly resolved. The developed software safety requirements shall become part of the software requirements specification. The software safety requirements shall be consistent with the precedence specified in Section 102.f of NHB 1700.1 (V1-B) and the Risk Assessment approach specified in Section 108 of NHB 1700.1 (V1-B).
The software safety requirements analysis shall follow the requirements given in Section 3.4.1.

3.3.1.3 SOFTWARE ARCHITECTURAL DESIGN

The software architectural design process develops the high-level design that will implement the software requirements. All software safety requirements developed in Section 3.3.1.2 shall be incorporated into the high-level software design as part of this process. The design process shall include identification of safety design features and methods (e.g., inhibits, traps, interlocks, and assertions) that will be used throughout the software to implement the software safety requirements. Safety-specific coding standards will also be developed that identify requirements for annotation of safety-critical code and limitation on use of certain language features that can reduce software safety. After allocation of the software safety requirements to the software design, component level Safety-Critical Computer Software Components (SCCSCs) shall be identified. These are all software components that implement software safety requirements or components that interface with SCCSCs that can affect their output.

During this phase, test planning to verify the correct implementation of the software safety requirements shall be completed. This planning shall include identification of tests that will be used to verify all software safety requirements and evaluate the correct response of the software to potential hazards. The safety activity shall review for concurrence the test plan.

Analysis shall be performed on the architectural design and test plan in accordance with Section 3.4.2.

3.3.1.4 SOFTWARE DETAILED DESIGN

The software detailed design process develops the low-level design for the software units that will implement the software requirements and high-level design. As part of the process, the high-level design for component level SCCSCs, including the safety design features and methods, shall be developed into a low level unit design. After development of the detail design, unit-level SCCSCs shall be identified. These SCCSCs are all software units that implement software safety requirements or units that interface with SCCSCs that can affect their output.

During this phase, safety-related information shall be incorporated into all user manuals. This information includes cautions, warnings, and procedures for handling safety-related procedures and hazards.

Test procedures that verify the software safety requirements shall be developed during this phase. The safety-related procedures shall include, but not be limited to, negative, no-go, off-nominal, and stress testing to ensure that the software responds correctly to hazards and does not initiate any hazards. These test procedures shall support Computer Software Configuration Item (CSCI), system and acceptance level testing. The safety activity shall review for concurrence the test procedures.

Analysis shall be performed on the detailed design to identify potential hazards and test procedures to ensure incorporation of safety-related testing in accordance with Section 3.4.3.

3.3.1.5 SOFTWARE IMPLEMENTATION

The software implementation translates the detailed design into code in the selected programming language. The code shall implement the safety design features and methods developed during the design process. Safety-critical code shall be commented in such a way that future changes can be made with a reduced likelihood of invoking a hazardous state. Analysis shall be performed on the code to identify potential hazards in accordance with Section 3.4.4.

The correct implementation of software safety requirements in the unit-level SCCSCs shall be verified to ensure accuracy and compliance with software engineering and safety detailed design requirements. Verification of each code unit must be completed prior to the unit's incorporation in the main code package.

Integration and acceptance test procedures that verify the software safety requirements shall be completed during this phase. The safety activity shall review for concurrence the completed test procedures.

Analysis shall be performed on test procedures to verify incorporation of safety-related testing in accordance with Section 3.4.4.

3.3.1.6 SOFTWARE INTEGRATION AND ACCEPTANCE TESTING

Testing shall be performed to verify correct incorporation of software safety requirements. Testing must show that hazards have been eliminated or controlled to an acceptable level of risk. Additional hazardous states identified during testing shall undergo complete analysis prior to software delivery or use. Software safety testing of SCCSCs shall be included in the integration and acceptance tests. Acceptance testing shall verify correct operation of the SCCSCs in conjunction with system hardware and operators. It shall verify correct operation during stress conditions and in the presence of system faults.

Analysis shall be performed on the test results in accordance with Section 3.4.5.

3.3.1.7 SOFTWARE OPERATIONS AND MAINTENANCE

The software safety processes defined in this section to specify, develop, analyze, and test SCCSCs shall be used when changes are made. The activities shall include: performing a hazard analysis, updating the software safety requirements; identifying new SCCSCs; updating the specification, design, and operator documentation for SCCSCs; updating and adding comments for safety-critical code; and testing the SCCSCs. Testing shall include regression testing to verify correct implementation of related software safety requirements.

3.3.2 PHASE INDEPENDENT TASKS

The following subsections describe those software safety tasks that are accomplished throughout the life cycle.

3.3.2.1 SAFETY REQUIREMENTS TRACEABILITY

A system shall be used to trace the flow down of the software safety requirements to design, implementation, and test. The tracing system shall also map the relationships between software safety requirements and system hazard reports.

3.3.2.2 DISCREPANCY REPORTING AND TRACKING

A system shall be used for closed-loop tracking of safety-related discrepancies, problems, and failures in baselined software products. All discrepancy reports shall be reviewed for safety impacts, with the safety activity's concurrence on safety-related discrepancy report closures. Analysis of changes made to correct discrepancies shall be performed in accordance with Section 3.4.6.

3.3.2.3 SOFTWARE CHANGE CONTROL

All changes, modifications, and patches made to the SCCSC requirements, design, code, systems, equipment, test plans, procedures, or criteria shall be evaluated to determine the effect of the proposed change on system/subsystem safety. The analysis shall be performed in accordance with Section 3.4.6.

3.3.2.4 SAFETY PROGRAM REVIEWS

Safety program reviews shall be conducted to ensure that implementation of safety controls of hazards are adequate. The software safety activity shall support the system safety review process.

3.4 SOFTWARE SAFETY ANALYSIS

The following sections provide a systematic approach for software safety analysis. They support the software safety process described in Section 3.3.

3.4.1 SOFTWARE SAFETY REQUIREMENTS ANALYSIS

A Software Safety Requirements Analysis (SSRA) shall be performed and documented. The system-level PHA and the system conceptual design shall be used as input to the

SSRA. The SSRA shall examine system-level software requirements, interface control documents, and the ongoing software requirements specification development to:

a. Identify software requirements that are safety critical.

b. Ensure the correctness and completeness of the decomposition of the high level safety requirements.

c. Provide safety-related recommendations for the design and testing process.

Analysis of all software requirements shall be performed in order to identify additional hazards that the system analysis did not include and to identify areas where system or interface requirements were not correctly assigned to the software. Identified hazards shall then be addressed by adding or changing the interfaces, system requirements, and/or software requirements. The SSRA shall consider such specific requirements as specific limit ranges; out-of-sequence event protection requirements (e.g., "if-then" statements); timers; relationship logic for interdependent limits; voting logic; hazardous command processing requirements; Fault Detection, Isolation, and Recovery (FDIR); and switchover logic for failure tolerance.

Output of the SSRA shall be used as input to follow-on software safety analyses. The SSRA shall be presented at the Software Requirements Review (SRR)/Software Specification Review (SSR) and system-level safety reviews. The results of the SSRA shall be provided to the ongoing system safety analysis activity.

3.4.2 SOFTWARE SAFETY ARCHITECTURAL DESIGN ANALYSIS

A Safety Architectural Design Analysis (SADA) shall be performed and documented. The Architectural Design, the results of the SSRA, and the system hazard analyses shall be used as inputs to the SADA. The SADA shall examine the software requirements specification, test plan, and the ongoing architectural design to:

a. Identify as SCCSCs those software components that implement the software safety requirements identified by the SSRA. Those software components that are found to affect the output of SCCSCs shall also be identified as SCCSCs.

b. Ensure the correctness and completeness of the architectural design as related to the software safety requirements and safety-related design recommendations.

c. Provide safety-related recommendations for the detailed design.

d. Ensure test coverage of the software safety requirements and provide recommendations for test procedures.

The output of the SADA shall be used as input to follow-on software safety analyses. The SADA shall be presented at the software Preliminary Design Review (PDR) and system-level safety reviews. The results of the SADA shall be provided to the ongoing system safety analysis activity.

3.4.3 SOFTWARE SAFETY DETAILED DESIGN ANALYSIS

A Safety Detailed Design Analysis (SDDA) shall be performed and documented. The Detailed Design, the results of the SSRA and SADA, and the system hazard analyses shall be used as inputs to the SDDA. The SDDA shall examine the software requirements specification, test procedures, and the ongoing detailed design to:

a. Refine the identification of SCCSCs to the unit level software components that implement the software safety requirements identified by the SSRA. Those unit-level software components that are found to affect the output of SCCSCs shall also be identified as SCCSCs.

b. Ensure the correctness and completeness of the detailed design as related to the software safety requirements, architectural design, and safety-related design recommendations.

c. Provide safety-related recommendations for code implementation.

d. Ensure test coverage of software safety requirements.

e. Develop safety-related information for inclusion in the User's Guide and other appropriate documentation.

The output of the SDDA shall be used as input to follow-on software safety analyses. The SDDA shall be presented at the software Critical Design Review (CDR) and system-level safety reviews. The results of the SDDA shall be provided to the ongoing system safety analysis activity.

3.4.4 CODE SAFETY ANALYSIS

A Code Safety Analysis (CSA) shall be performed and documented. The code, the results of the SSRA, SADA, and SDDA, and the system hazard analyses shall be used as inputs to the CSA. The CSA shall examine the software requirements specification, test procedures, and the ongoing code development to:

a. Ensure the correctness and completeness of the code as related to the software safety requirements, detailed design, and safety-related coding recommendations.

b. Identify potentially unsafe states caused by input/output timing, multiple events, out-of-sequence events, failure of events, adverse environments, deadlocking, wrong events, inappropriate magnitude, improper polarity, and hardware failure sensitivities, etc.

c. Ensure that SCCSCs are adequately commented.

d. Ensure test coverage of software safety requirements

e. Update safety-related information for inclusion in the User's Guide and other appropriate documentation.

The status of the software safety analysis shall be presented at the Test Readiness Review (TRR). The results of the CSA shall be provided to the ongoing system safety analysis activity.

3.4.5 SOFTWARE TEST SAFETY ANALYSIS

The test results shall be analyzed to verify that all safety requirements have been satisfied. The analysis shall also verify that all identified hazards have been eliminated or controlled to an acceptable level of risk. The results of the test safety analysis shall be provided to the ongoing system safety analysis activity.

3.4.6 SOFTWARE CHANGE ANALYSIS

Software change analysis shall evaluate whether the proposed change could invoke a hazardous state, affect a hazard control, increase the likelihood of a hazardous state, adversely affect safety-critical software, or change the criticality of a software component. The analysis shall also ensure that any affected documentation is updated to correctly reflect any safety-related changes that have been made.

This page intentionally left blank.

4.0 QUALITY ASSURANCE PROVISIONS

Quality Assurance (QA) shall assure:

a. Software safety planning is performed, approved, and implemented.

b. Technical recommendations resulting from software safety activities are reviewed, considered by change control authority, and where appropriate, implemented.

c. Reviews and audits address software safety concerns, requirements, and guidelines.

d. Software safety processes, product standards, and procedures are followed and met.

QA shall observe key elements of tasks performed to verify proper execution of approved plans and procedures.

This page intentionally left blank.

5.0 PACKAGING

This section is not applicable to this standard.

This page intentionally left blank.

6.0 ADDITIONAL INFORMATION

This section is not applicable to this standard.

This page intentionally left blank.

APPENDIX A

GLOSSARY

Various definitions contained in this Glossary are reproduced from IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology, copyright ©1990 by the Institute of Electrical and Electronic Engineers, Inc. The IEEE takes no responsibility for and will assume no liability for damages resulting from the reader's misinterpretation of said information resulting from the placement and context in this publication. Information is reproduced with the permission of the IEEE.

Assertions. A logical expression specifying a program state that must exist or a set of conditions that program variables must satisfy at a particular point during a program execution. Types include input assertion, loop assertion, and output assertion. (IEEE Standard 610.12-1990)

Code Safety Analysis (CSA). An analysis of program code and system interfaces for events, faults, and conditions that could cause or contribute to undesirable events affecting safety.

Command. Any message that causes the receiving party to perform an action.

Computer Software Configuration Item (CSCI). An aggregate of software that is designated for configuration management and is treated as a single entity in the configuration management process. (IEEE Standard 610.12-1990)

Concept/Conceptual. The period of time in the software development cycle during which the user needs are described and evaluated through documentation (for example, statement of needs, advance planning report, project initiation memo, feasibility studies, system definition, documentation, regulations, procedures, or policies relevant to the project).

Critical Design Review (CDR). A review conducted to verify that the detailed design of one or more configuration items satisfy specified requirements; to establish the compatibility among configuration items and other items of equipment, facilities, software, and personnel; to assess risk areas for each configuration item; and, as applicable, to assess the results of the producibility analyses, review preliminary hardware product specifications, evaluate preliminary test planning, and evaluate the adequacy of preliminary operation and support documents. (IEEE Standard 610.12-1990)

For Computer Software Configuration Items (CSCIs), this review will focus on the determination of the acceptability of the detailed design, performance, and test characteristics of the design solution, and on the adequacy of the operation and support documents.

Deadlock. A situation in which computer processing is suspended because two or more devices or processes are each awaiting resources assigned to the other. (IEEE Standard 610.12-1990)

Failure. The inability of a system or component to perform its required functions within specified performance requirements. (IEEE Standard 610.12-1990)

Failure Tolerance. The ability of a system or subsystem to perform its function(s) or maintain control of a hazard in the presence of failures within its hardware, firmware, or software.

Fault. Any change in state of an item that is considered to be anomalous and may warrant some type of corrective action. Examples of faults included device errors reported by Built-In Test (BIT)/Built-In Test Equipment (BITE), out-of-limits conditions on sensor values, loss of communication with devices, loss of power to a device, communication error on bus transaction, software exceptions (e.g., divide by zero, file not found), rejected commands, measured performance values outside of commanded or expected values, an incorrect step, process, or data definition in a computer program, etc. Faults are preliminary indications that a failure may have occurred.

Fault Detection. A process that discovers or is designed to discover faults; the process of determining that a fault has occurred.

Fault Isolation. The process of determining the location or source of a fault.

Fault Recovery. A process of elimination of a fault without permanent reconfiguration.

Firmware. Computer programs and data loaded in a class of memory that cannot be dynamically modified by the computer during processing.

Hazard. Existing or potential condition that can result in or contribute to a mishap.

Hazardous Command. A command whose execution (including inadvertent, out-of-sequence, or incorrectly executed) could lead to an identified critical or catastrophic hazard, or a command whose executions can lead to a reduction in the control of a hazard (including reduction in failure tolerance against a hazard or the elimination of an inhibit against a hazard).

Independent Verification and Validation (IV&V). A process whereby the products of the software development life cycle phases are independently reviewed, verified, and validated by an organization that represents the acquirer of the software and is completely independent of the provider.

Inhibit. A design feature that provides a physical interruption between an energy source and a function (e.g., a relay or transistor between a battery and a pyrotechnic initiator, a latch valve between a propellant tank and a thruster, etc.).

Interlock. Hardware or software function that prevents succeeding operations when specific conditions exist.

Life Cycle. The period of time that starts when a software product is conceived and ends when the software is no longer available for use. The software life cycle traditionally has eight phases: Concept and Initiation; Requirements; Architectural Design; Detailed Design; Implementation; Integration and Test; Acceptance and Delivery; and Sustaining Engineering and Operations.

Mishap. An unplanned event or series of events that results in death, injury, occupational illness, or damage to or loss of equipment, property, or damage to the environment; an accident.

Negative Testing. Software Safety Testing to ensure that the software will not go to a hazardous state or generate outputs that will create a hazard in the system in response to out of bound or illegal inputs.

No-Go Testing. Software Safety Testing to ensure that the software performs known processing and will go to a known safe state in response to specific hazardous situations.

Preliminary Design Review (PDR). A review conducted to evaluate the progress, technical adequacy, and risk resolution of the selected design approach for one or more configuration items; to determine each design's compatibility with the requirements for the configuration item; to evaluate the degree of definition and assess the technical risk associated with the selected manufacturing methods and processes; to establish the existence and compatibility of the physical and functional interfaces among the configuration items and other items of equipment, facilities, software, and personnel; and as appropriate, to evaluate the preliminary operation and support documents. (IEEE Standard 610.12-1990)

For CSCIs, the review will focus on: (1) the evaluation of the progress, consistency, and technical adequacy of the selected architectural design and test approach, (2) compatibility between software requirements and architectural design, and (3) the preliminary version of the operation and support documents.

Preliminary Hazard Analysis (PHA). Analysis performed at the system level to identify safety-critical areas, to provide an initial assessment of hazards, and to identify requisite hazard controls and follow-on actions.

Risk. As it applies to safety, exposure to the chance of injury or loss. It is a function of the possible frequency of occurrence of the undesired event, of the potential severity of resulting consequences, and of the uncertainties associated with the frequency and severity.

Safety Analysis. A systematic and orderly process for the acquisition and evaluation of specific information pertaining to the safety of a system.

Safety Architectural Design Analysis (SADA). Analysis performed on the high-level design to verify the correct incorporation of safety requirements and to analyze the Safety-Critical Computer Software Components (SCCSCs).

Safety-Critical. Those software operations that, if not performed, performed out-of sequence, or performed incorrectly could result in improper control functions (or lack of control functions required for proper system operation) that could directly or indirectly cause or allow a hazardous condition to exist.

Safety-Critical Computer Software Component (SCCSC). Those computer software components (processes, modules, functions, values or computer program states) whose errors (inadvertent or unauthorized occurrence, failure to occur when required, occurrence out of sequence, occurrence in combination with other functions, or erroneous value) can result in a potential hazard, or loss of predictability or control of a system.

Safety-Critical Software. Software that: (1) Exercises direct command and control over the condition or state of hardware components; and, if not performed, performed out-of-sequence, or performed incorrectly could result in improper control functions (or lack of control functions required for proper system operation), which could cause a hazard or allow a hazardous condition to exist. (2) Monitors the state of hardware components; and, if not performed, performed out-of-sequence, or performed incorrectly could provide data that results in erroneous decisions by human operators or companion systems that could cause a hazard or allow a hazardous condition to exist. (3) Exercises direct command and control over the condition or state of hardware components; and, if performed inadvertently, out-of-sequence, or if not performed, could, in conjunction with other human, hardware, or environmental failure, cause a hazard or allow a hazardous condition to exist.

Safety Detailed Design Analysis (SDDA). Analysis performed on Safety-Critical Computer Software Components to verify the correct incorporation of safety requirements and to identify additional hazardous conditions.

Software Requirements Review (SRR). A review of the requirements specified for one or more software configuration items to evaluate their responsiveness to and interpretation of system requirements and to determine whether they form a satisfactory basis for proceeding into a preliminary (architectural) design of configuration items. (IEEE Standard 610.12-1990)

Same as Software Specification Review for DoD-STD-2167A.

Software Requirements Specification (SRS). Documentation of the essential requirements (functions, performance, design constraints, and attributes) of the software and its external interfaces. (IEEE Standard 610.12-1990)

Software Safety Requirements Analysis (SSRA). Analysis performed to examine system and software requirements and the conceptual design in order to identify unsafe modes for resolution, such as out-of-sequence, wrong event, deadlocking, and failure-to-command modes.

Software Specification Review (SSR). Same as Software Requirements Review.

Software Safety. The application of the disciplines of system safety engineering techniques throughout the software life cycle to ensure that the software takes positive measures to enhance system safety and that errors that could reduce system safety have been eliminated or controlled to an acceptable level of risk.

System Safety. Application of engineering and management principles, criteria, and techniques to optimize safety and reduce risks within the constraints of operational effectiveness, time, and cost throughout all phases of the system life cycle.

Test Readiness Review (TRR). A review conducted to evaluate preliminary test results for one or more configuration items; to verify that the test procedures for each configuration item are complete, comply with test plans and descriptions, and satisfy test requirements; and to verify that a project is prepared to proceed to formal test of the configuration items. (IEEE Standard 610.12-1990)

Trap. Software feature that monitors program execution and critical signals to provide additional checks over and above normal program logic. Traps provide protection against undetected software errors, hardware faults, and unexpected hazardous conditions.

Validation. (1) An evaluation technique to support or corroborate safety requirements to ensure necessary functions are complete and traceable. (2) The process of evaluating software at the end of the software development process to ensure compliance with software requirements.

Verification. (1) The process of determining whether the products of a given phase of the software development cycle fulfill the requirements established during the previous phase (see also validation). (2) Formal proof of program correctness. (3) The act of reviewing, inspecting, testing, checking, auditing, or otherwise establishing and documenting whether items, processes, services, or documents conform to specified requirements.

Waiver. A variance that authorizes departure from a particular safety requirement where alternate methods are employed to mitigate risk or where an increased level of risk has been accepted by management.

This page intentionally left blank.

APPENDIX B

ABBREVIATIONS AND ACRONYMS

BIT	Built-In Test
BITE	Built-In Test Equipment
CDR	Critical Design Review
COTS	Commercial Off-the-Shelf
CSA	Code Safety Analysis
CSCI	Computer Software Configuration Item
DID	Data Item Description
DoD	Department of Defense
FDIR	Fault Detection, Isolation, and Recovery
GFE	Government Furnished Equipment
IEEE	Institute of Electrical and Electronic Engineers
IV&V	Independent Verification and Validation
MIL-STD	Military Standard
NHB	NASA Handbook
NMI	NASA Management Instruction
QA	Quality Assurance
PDR	Preliminary Design Review
PHA	Preliminary Hazard Analysis
SADA	Safety Architectural Design Analysis
SCCSC	Safety-Critical Computer Software Component
SDDA	Safety Detailed Design Analysis
SSRA	Software Safety Requirements Analysis
SRR	Software Requirements Review
SRS	Software Requirements Specification
SSR	Software Specification Review
TRR	Test Readiness Review

This page intentionally left blank.